Foundations of Software
Fall Semester 2009

Week 7

Plan

PREVIOUSLY: unit, sequencing, let, pairs, tuples

TODAY:

1. options, variants
2. recursion
3. state

NEXT: exceptions?
NEXT: polymorphic (not so simple) typing

Records

t	::=	...	*terms*
		$\{l_i = t_i\ ^{i \in 1..n}\}$	*record*
		t.l	*projection*
v	::=	...	*values*
		$\{l_i = v_i\ ^{i \in 1..n}\}$	*record value*
T	::=	...	*types*
		$\{l_i : T_i\ ^{i \in 1..n}\}$	*type of records*

Evaluation rules for records

$$\{l_i = v_i\ ^{i \in 1..n}\}.l_j \longrightarrow v_j \qquad \text{(E-ProjRcd)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.l \longrightarrow t_1'.l} \qquad \text{(E-Proj)}$$

$$\frac{t_j \longrightarrow t_j'}{\begin{array}{l}\{l_i = v_i\ ^{i \in 1..j-1}, l_j = t_j, l_k = t_k\ ^{k \in j+1..n}\} \\ \longrightarrow \{l_i = v_i\ ^{i \in 1..j-1}, l_j = t_j', l_k = t_k\ ^{k \in j+1..n}\}\end{array}} \qquad \text{(E-Rcd)}$$

Typing rules for records

$$\frac{\text{for each } i \quad \Gamma \vdash \mathtt{t}_i : \mathtt{T}_i}{\Gamma \vdash \{\mathtt{l}_i\mathtt{=}\mathtt{t}_i\ ^{i \in 1..n}\} : \{\mathtt{l}_i\mathtt{:}\mathtt{T}_i\ ^{i \in 1..n}\}} \quad \text{(T-Rcd)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \{\mathtt{l}_i\mathtt{:}\mathtt{T}_i\ ^{i \in 1..n}\}}{\Gamma \vdash \mathtt{t}_1.\mathtt{l}_j : \mathtt{T}_j} \quad \text{(T-Proj)}$$

Sums and variants

Sums – motivating example

```
PhysicalAddr = {firstlast:String, addr:String}
VirtualAddr  = {name:String, email:String}
Addr         = PhysicalAddr + VirtualAddr
inl  :  "PhysicalAddr → PhysicalAddr+VirtualAddr"
inr  :  "VirtualAddr → PhysicalAddr+VirtualAddr"
```

```
getName = λa:Addr.
  case a of
    inl x ⇒ x.firstlast
  | inr y ⇒ y.name;
```

New syntactic forms

t	::=	...	*terms*
		inl t	*tagging (left)*
		inr t	*tagging (right)*
		case t of inl x⇒t | inr x⇒t	*case*
v	::=	...	*values*
		inl v	*tagged value (left)*
		inr v	*tagged value (right)*
T	::=	...	*types*
		T+T	*sum type*

$\mathtt{T}_1\mathtt{+}\mathtt{T}_2$ is a *disjoint union* of $\mathtt{T}_1$ and $\mathtt{T}_2$ (the tags `inl` and `inr` ensure disjointness)

New evaluation rules $\boxed{t \longrightarrow t'}$

$$\texttt{case (inl } v_0\texttt{) of inl } x_1 \Rightarrow t_1 \mid \texttt{inr } x_2 \Rightarrow t_2 \longrightarrow [x_1 \mapsto v_0]t_1 \quad \text{(E-CaseInl)}$$

$$\texttt{case (inr } v_0\texttt{) of inl } x_1 \Rightarrow t_1 \mid \texttt{inr } x_2 \Rightarrow t_2 \longrightarrow [x_2 \mapsto v_0]t_2 \quad \text{(E-CaseInr)}$$

$$\frac{t_0 \longrightarrow t_0'}{\texttt{case } t_0 \texttt{ of inl } x_1 \Rightarrow t_1 \mid \texttt{inr } x_2 \Rightarrow t_2 \longrightarrow \texttt{case } t_0' \texttt{ of inl } x_1 \Rightarrow t_1 \mid \texttt{inr } x_2 \Rightarrow t_2} \quad \text{(E-Case)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{inl } t_1 \longrightarrow \texttt{inl } t_1'} \quad \text{(E-Inl)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{inr } t_1 \longrightarrow \texttt{inr } t_1'} \quad \text{(E-Inr)}$$

New typing rules $\boxed{\Gamma \vdash t : T}$

$$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \texttt{inl } t_1 : T_1 + T_2} \quad \text{(T-Inl)}$$

$$\frac{\Gamma \vdash t_1 : T_2}{\Gamma \vdash \texttt{inr } t_1 : T_1 + T_2} \quad \text{(T-Inr)}$$

$$\frac{\Gamma \vdash t_0 : T_1 + T_2 \qquad \Gamma, x_1{:}T_1 \vdash t_1 : T \qquad \Gamma, x_2{:}T_2 \vdash t_2 : T}{\Gamma \vdash \texttt{case } t_0 \texttt{ of inl } x_1 \Rightarrow t_1 \mid \texttt{inr } x_2 \Rightarrow t_2 : T} \quad \text{(T-Case)}$$

Sums and Uniqueness of Types

Problem:

If `t` *has type* `T`*, then* `inl t` *has type* `T+U` *for every* `U`*.*

I.e., we've lost uniqueness of types.

Possible solutions:

- "Infer" `U` as needed during typechecking
- Give constructors different names and only allow each name to appear in one sum type (requires generalization to "variants," which we'll see next) — OCaml's solution
- Annotate each `inl` and `inr` with the intended sum type.

For simplicity, let's choose the third.

New syntactic forms

`t ::=`	...	*terms*
	`inl t as T`	*tagging (left)*
	`inr t as T`	*tagging (right)*
`v ::=`	...	*values*
	`inl v as T`	*tagged value (left)*
	`inr v as T`	*tagged value (right)*

Note that `as T` here is not the ascription operator that we saw before — i.e., not a separate syntactic form: in essence, there is an ascription "built into" every use of `inl` or `inr`.

New typing rules $\boxed{\Gamma \vdash t : T}$

$$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \texttt{inl } t_1 \texttt{ as } T_1\texttt{+}T_2 : T_1\texttt{+}T_2} \quad \text{(T-Inl)}$$

$$\frac{\Gamma \vdash t_1 : T_2}{\Gamma \vdash \texttt{inr } t_1 \texttt{ as } T_1\texttt{+}T_2 : T_1\texttt{+}T_2} \quad \text{(T-Inr)}$$

Evaluation rules ignore annotations: $\boxed{t \longrightarrow t'}$

$$\begin{array}{l}\texttt{case (inl } v_0 \texttt{ as } T_0\texttt{)} \\ \texttt{of inl } x_1 \Rightarrow t_1 \mid \texttt{inr } x_2 \Rightarrow t_2 \\ \qquad \longrightarrow [x_1 \mapsto v_0]t_1\end{array} \quad \text{(E-CaseInl)}$$

$$\begin{array}{l}\texttt{case (inr } v_0 \texttt{ as } T_0\texttt{)} \\ \texttt{of inl } x_1 \Rightarrow t_1 \mid \texttt{inr } x_2 \Rightarrow t_2 \\ \qquad \longrightarrow [x_2 \mapsto v_0]t_2\end{array} \quad \text{(E-CaseInr)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{inl } t_1 \texttt{ as } T_2 \longrightarrow \texttt{inl } t_1' \texttt{ as } T_2} \quad \text{(E-Inl)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{inr } t_1 \texttt{ as } T_2 \longrightarrow \texttt{inr } t_1' \texttt{ as } T_2} \quad \text{(E-Inr)}$$

Variants

Just as we generalized binary products to labeled records, we can generalize binary sums to labeled *variants*.

New syntactic forms

t	::=	...	*terms*
		$\texttt{<l=t> as T}$	*tagging*
		$\texttt{case t of <}l_i\texttt{=}x_i\texttt{>} \Rightarrow t_i \ ^{i \in 1..n}$	*case*
T	::=	...	*types*
		$\texttt{<}l_i\texttt{:}T_i \ ^{i \in 1..n}\texttt{>}$	*type of variants*

New evaluation rules $\boxed{\texttt{t} \longrightarrow \texttt{t}'}$

$$\texttt{case } (\texttt{<l}_j\texttt{=v}_j\texttt{>} \texttt{ as T}) \texttt{ of } \texttt{<l}_i\texttt{=x}_i\texttt{>}\Rightarrow\texttt{t}_i \;^{i\in 1..n} \longrightarrow [\texttt{x}_j \mapsto \texttt{v}_j]\texttt{t}_j \qquad \text{(E-CaseVariant)}$$

$$\frac{\texttt{t}_0 \longrightarrow \texttt{t}_0'}{\texttt{case t}_0 \texttt{ of <l}_i\texttt{=x}_i\texttt{>}\Rightarrow\texttt{t}_i \;^{i\in 1..n} \longrightarrow \texttt{case t}_0' \texttt{ of <l}_i\texttt{=x}_i\texttt{>}\Rightarrow\texttt{t}_i \;^{i\in 1..n}} \qquad \text{(E-Case)}$$

$$\frac{\texttt{t}_i \longrightarrow \texttt{t}_i'}{\texttt{<l}_i\texttt{=t}_i\texttt{> as T} \longrightarrow \texttt{<l}_i\texttt{=t}_i'\texttt{> as T}} \qquad \text{(E-Variant)}$$

New typing rules $\boxed{\Gamma \vdash \texttt{t} : \texttt{T}}$

$$\frac{\Gamma \vdash \texttt{t}_j : \texttt{T}_j}{\Gamma \vdash \texttt{<l}_j\texttt{=t}_j\texttt{> as <l}_i\texttt{:T}_i \;^{i\in 1..n}\texttt{>} : \texttt{<l}_i\texttt{:T}_i \;^{i\in 1..n}\texttt{>}} \qquad \text{(T-Variant)}$$

$$\frac{\Gamma \vdash \texttt{t}_0 : \texttt{<l}_i\texttt{:T}_i \;^{i\in 1..n}\texttt{>} \qquad \text{for each } i \quad \Gamma, \texttt{x}_i\texttt{:T}_i \vdash \texttt{t}_i : \texttt{T}}{\Gamma \vdash \texttt{case t}_0 \texttt{ of <l}_i\texttt{=x}_i\texttt{>}\Rightarrow\texttt{t}_i \;^{i\in 1..n} : \texttt{T}} \qquad \text{(T-Case)}$$

Example

```
Addr = <physical:PhysicalAddr, virtual:VirtualAddr>;

a = <physical=pa> as Addr;

getName = λa:Addr.
  case a of
    <physical=x> ⇒ x.firstlast
  | <virtual=y> ⇒ y.name;
```

Options

Just like in OCaml...

```
OptionalNat = <none:Unit, some:Nat>;

Table = Nat→OptionalNat;

emptyTable = λn:Nat. <none=unit> as OptionalNat;

extendTable =
  λt:Table. λm:Nat. λv:Nat.
    λn:Nat.
      if equal n m then <some=v> as OptionalNat
      else t n;

x = case t(5) of
      <none=u> ⇒ 999
    | <some=v> ⇒ v;
```

Enumerations

```
Weekday = <monday:Unit, tuesday:Unit, wednesday:Unit,
           thursday:Unit, friday:Unit>;

nextBusinessDay = λw:Weekday.
  case w of <monday=x>    ⇒ <tuesday=unit> as Weekday
          | <tuesday=x>   ⇒ <wednesday=unit> as Weekday
          | <wednesday=x> ⇒ <thursday=unit> as Weekday
          | <thursday=x>  ⇒ <friday=unit> as Weekday
          | <friday=x>    ⇒ <monday=unit> as Weekday;
```